GRAPHICAL FISHEYE VIEWS OF GRAPHS

*Manojit Sarkar**
Department of Computer Science
Brown University
Providence, RI 02912
ms@cs.brown.edu

Marc H. Brown
Systems Research Center
Digital Equipment Corporation
Palo Alto, CA 94301
mhb@src.dec.com

ABSTRACT

A *fisheye* lens is a very wide angle lens that shows places nearby in detail while also showing remote regions in successively less detail. This paper describes a system for viewing and browsing planar graphs using a software analog of a fisheye lens. We first show how to implement such a view using solely geometric transformations. We then describe a more general transformation that allows hierarchical, structured information about the graph to modify the views. Our general transformation is a fundamental extension to the previous research in fisheye views.

KEYWORDS: Fisheye views, information visualization

INTRODUCTION

Graphs with hundreds of vertices and edges are common in many areas of computer science, such as network topology, VLSI circuits, and graph theory. There are literally hundreds of algorithms for positioning nodes to produce an aesthetic and informative display [1]. However, once a layout is chosen, what is an effective way to view and browse the graph on a workstation?

Displaying all the information associated with the vertices and edges (assuming it can even fit on a screen) shows the global structure of the graph, but has the drawback that details are typically too small to be seen. Alternatively, zooming into a part of the graph and panning to other parts does show local details but loses the overall structure of the graph. Researchers have found that browsing a large layout by scrolling and arc traversing tends to obscure the global structure of the graph [6]. Two (or more) views — one view of the entire graph and the other of a zoomed portion — has the advantage of seeing both the local detail and overall structure, but has the drawbacks of requiring extra screen space and of forcing the viewer to mentally integrate the views. The multiple view approach also has the drawback that parts of the graph adjacent to the enlarged area are not visible at all in the enlarged view.

*Supported by summer research internship from Digital Equipment Corporation's Systems Research Center.

This paper explores a *fisheye* lens approach to viewing and browsing graphs. A fisheye view of a graph shows an area of interest quite large and with detail, and shows the remainder of the graph successively smaller and in less detail. Thus, a fisheye lens seems to have all the advantages of the other approaches and without suffering from any of the drawbacks.

A typical graph is displayed in Figure 1, and a fisheye version of it appears in Figure 2. In the fisheye view, the vertex with thick border is the current point of interest to the viewer. We call this point the *focus*. In our prototype system, a viewer selects the focus by clicking with a mouse. As the mouse is dragged, the focus changes and the display updates in real time. The size and detail of a vertex in the fisheye view depend on the distance of the vertex from the focus, a preassigned importance associated with the vertex, and the values of some user-controlled parameters.

Our work extends Furnas's pioneering work on fisheye views [4, 5] by providing a graphical interpretation to fisheye views. We introduce layout considerations into the fisheye formalism, so that the position, size, and level of detail of objects displayed are computed based on client-specified functions of an object's distance from the focus and the object's preassigned importance in the global structure. In Furnas's original formulation of the fisheye view, a component is either present in full detail or is completely absent from the view, and there is no explicit control over the graphical layout.

TERMINOLOGY

A graph consists of *vertices* and *edges*. The initial layout of the graph is called the *normal view* of the graph, and its coordinates are called *normal coordinates*. Vertices are graphically represented by shapes whose bounding boxes are square (chosen arbitrarily). Each vertex has a *position*, specified by its normal coordinates, and a *size* which is the

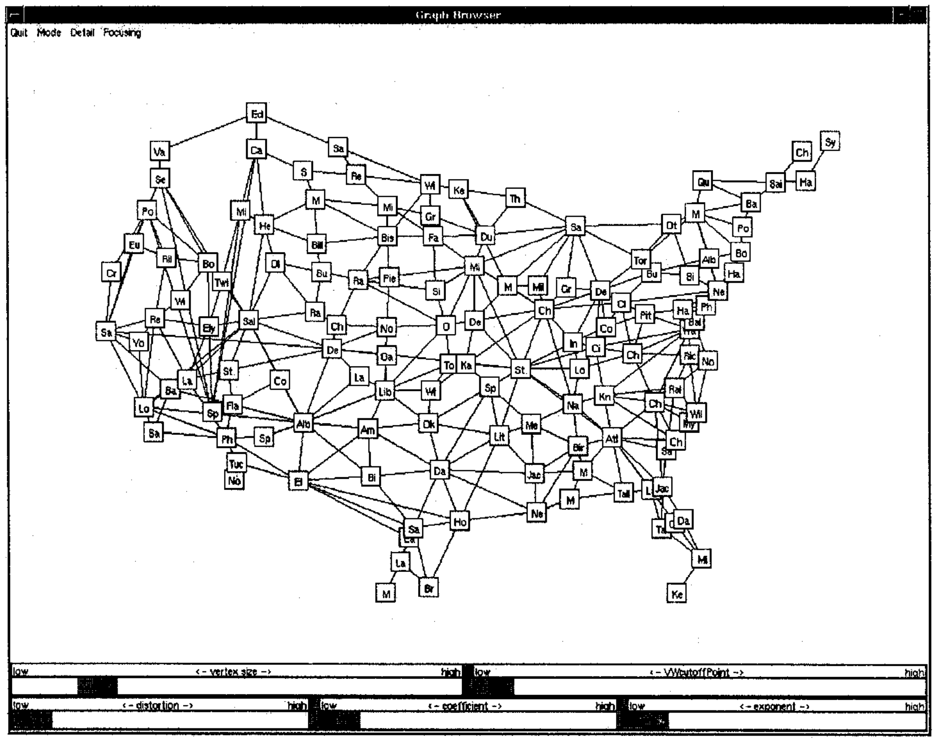

Graph Browser
Quit Mode Detail Focusing
low <- vertex size -> high
low <- VWcutoffPoint -> high
low <- distortion -> high
low <- coefficient -> high
low <- exponent -> high

Figure 1: The initial layout of a graph with 134 vertices and 338 edges.

length of a side of the bounding box of the vertex. Each vertex is also assigned a number to represent its relative importance in the global structure. This number is called the *a priori importance*, or the *API*, of the vertex.

An edge is represented by either a straight line from one vertex to another, or by a set of straight line segments to simulate curved edges. Edges consisting of multiple straight line segments are specified by a set of intermediate *bend points*, the extreme points being the coordinates of its corresponding vertices.

The coordinates of the graph in the fisheye view are called the *fisheye coordinates*. The viewer's point of interest is called the *focus*; it is a point in the normal coordinates. Each vertex is the fisheye view is defined by its position, size, and the *amount of detail* to display. Finally, each vertex in fisheye view is assigned a *visual worth*, or *VW*, computed based on its distance to the focus (in normal coordinates) and its *a priori* importance.

GENERATING FISHEYE VIEWS

Generating a fisheye view involves magnifying the vertices of greater interest and correspondingly demagnifying the vertices of lower interest. In addition, the positions of all vertices and bend points must also be recomputed in order to allocate more space for the magnified portion so that the entire view still occupies the same amount of screen space.

Intuitively, the position of a vertex in the fisheye view depends on its position in the normal view and its distance from the focus. The size of a vertex in the fisheye view depends on its distance from the focus, its size in the normal view, and its API. The amount of detail displayed in a vertex in turn depends on its size in the fisheye view. We now formalize these concepts.

The position of vertex v in the fisheye view is a function of its position in normal coordinates and the position of the focus:

$$P_{feye}(v,f) = \mathcal{F}_1(P_{norm}(v), P_{norm}(f)) \qquad (1)$$

The size of vertex v in the fisheye view is a function of its size and position in normal coordinates, the position of the focus, and its API:

$$S_{feye}(v,f) = \mathcal{F}_2(S_{norm}(v), P_{norm}(v), P_{norm}(f), API(v)) \qquad (2)$$

The amount of detail to be shown for vertex v depends on the size of v in the fisheye view and the maximum detail that can be displayed:

$$DTL_{feye}(v,f) = \mathcal{F}_3(S_{feye}(v,f), DTL_{maximum}(v)) \qquad (3)$$

Finally, the visual worth of vertex v depends on the distance between v and the focus in normal coordinates and on v's API:

$$VW(v,f) = \mathcal{F}_4(D_{norm}(v,f), API(v)) \qquad (4)$$

One has to choose the functions $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4$ appropriately to generate useful fisheye views. Readers familiar with Furnas's work will note that our fundamental contributions are the existence of arbitrary functions $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_3$. In

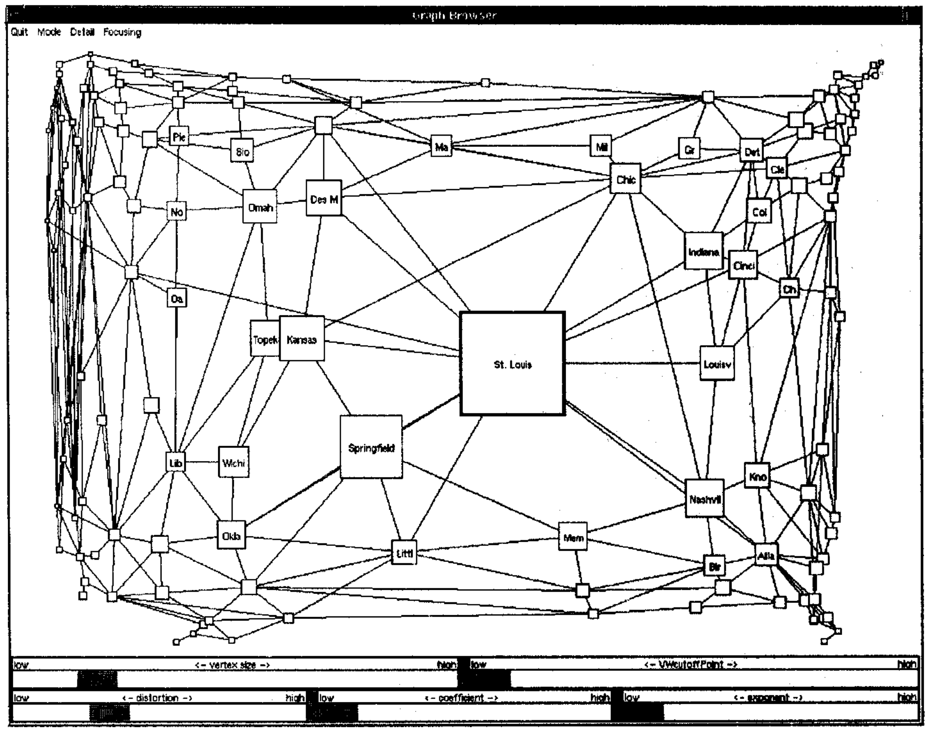


Figure 2: A fisheye view of the graph in Figure 1, $d = 5.71, c = 0, e = 0, \mathit{VWcutoff} = 0$

the next section, we present the set of functions we used in our prototype system.

FISHEYE TRANSFORMATIONS

Generating fisheye views is a two step process. First we apply a geometric transformation to the normal view in order to reposition vertices and magnify and demagnify areas close to and far away from the focus, respectively. Second, we use the API of vertices to obtain their final size, detail, and visual worth. In some applications, the API of all the vertices are equal, so the final size of all the vertices are equal and the second step is therefore unnecessary.

Mapping Position

Transforming from normal coordinates to fisheye coordinates, using focus position P_{focus} requires us to implement the function $\mathcal{F}_1$ in Equation 1. The function we used was

$$P_{feye} = \mathcal{G}(P_{norm})D_{max} + P_{focus}$$

where

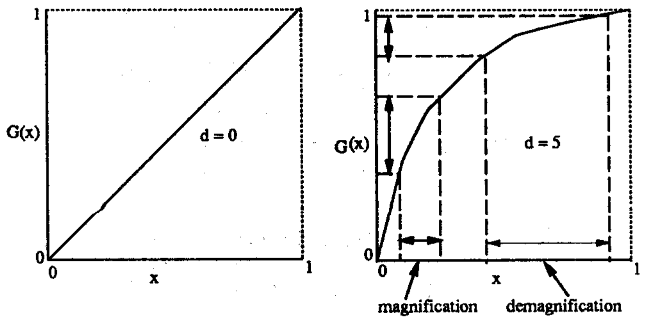
$$\mathcal{G}(P_{norm}) = \frac{(d+1)\frac{D_{norm}}{D_{max}}}{d\frac{D_{norm}}{D_{max}} + 1} = \frac{d+1}{d + \frac{D_{max}}{D_{norm}}} \qquad (5)$$

Note that the x and y dimensions are treated completely independently in the above mapping. This mapping is called the *cartesian* transformation. Later, we show a slightly different transformation called the *polar* transformation which is based on the polar coordinate system.

D_{max} is the distance of the boundary of the screen from the focus. The constant d in Equation 5 is called the *distortion factor*. The function $\mathcal{G}(x)$ is monotonically increasing and continuous for $0 \leq x \leq 1$ with $\mathcal{G}(0) = 0$, and $\mathcal{G}(1) = 1$. The derivative of $\mathcal{G}(x)$ is

$$\mathcal{G}'(x) = \frac{d+1}{(dx+1)^2} \qquad (6)$$

This indicates that for large values of d the slope of the plot x versus $\mathcal{G}(x)$ near $x = 0$ is very high. This results in high magnification. The plot has a very low slope near $x = 1$ which causes high demagnification. The behavior of the function for $d = 0$ and $d = 5$ is as follows:



When $d = 0$, the normal and the fisheye coordinates of every point are the same.

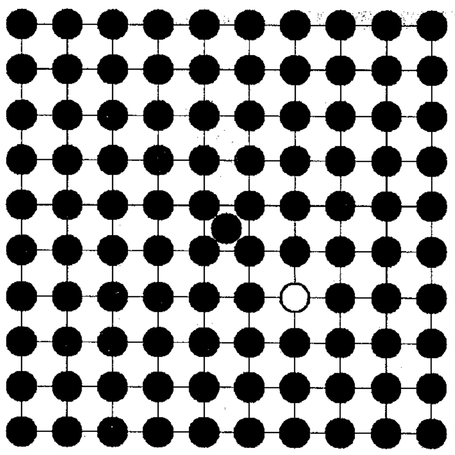


Figure 3: An undistorted symmetric graph, $d = 0$

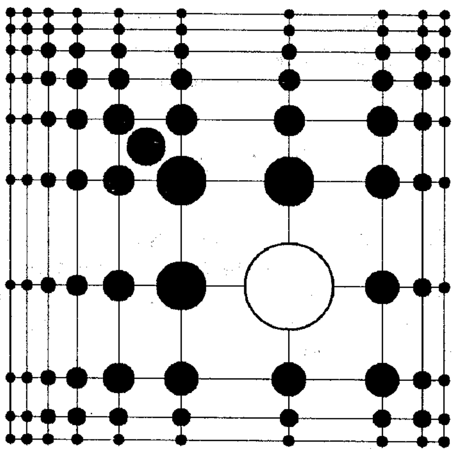


Figure 4: Cartesian transformation of symmetric graph, $d = 4$

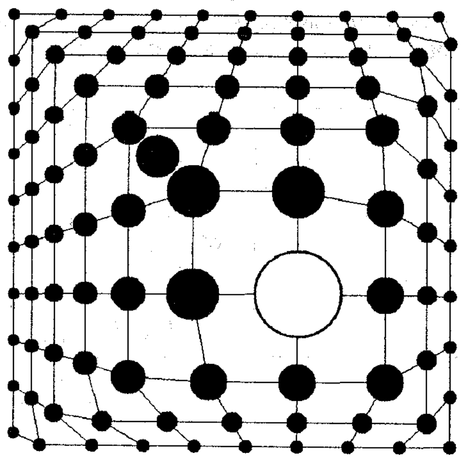


Figure 5: Polar transformation of symmetric graph, $d = 4$

Mapping Size

While computing size, the square shape of the bounding boxes of the vertices is preserved. Implementation of the size mapping function $\mathcal{F}_2$ in Equation 2 is described here in two separate steps. The first step uses the geometric transformation just found in order to compute the geometric size $S_{geom}(v, f)$ by ignoring v's API. This mapping has the special property that if no two vertices in the normal view overlapped, no two vertices in the transformed view overlap. The second step then uses $S_{geom}(v, f)$ and v's API to complete the implementation of $\mathcal{F}_2$. However, the vertices may overlap after the second step. We compute S_{geom} as follows:

$$S_{geom} = \sqrt{2}\min(MapX(x_{norm} + \frac{S_{norm}}{2}) - x_{feye}, \quad MapY(y_{norm} + \frac{S_{norm}}{2}) - y_{feye})$$

where $P_{norm} = (x_{norm}, y_{norm})$ and $P_{feye} = (x_{feye}, y_{feye})$. The functions *MapX* and *MapY* map the x and y coordinates of a point respectively using the function $\mathcal{F}_1$ in Equation 1. This equation can be derived by taking definite integral of Equation 6 on x and y dimensions independently with appropriate limits, and then by making other necessary adjustments.

Finally, the function $\mathcal{F}_2$ in Equation 2 is implemented by

$$S_{feye} = S_{geom}(c \cdot API)^e s \tag{7}$$

where the coefficient c, exponent e, and scale factor s are constants.

Computing Detail and Visual Worth

The functions $\mathcal{F}_3$ and $\mathcal{F}_4$ are implemented by Equation 8 and Equation 9 below. Both equations utilize S_{feye} computed by Equation 7.

$$DTL_{feye}(v, f) = \min(DTL_{maximum}(v), \alpha S_{feye}(v, f)) \tag{8}$$

where α is a constant.

$$VW(v, f) = \beta S_{feye}(v, f) + \gamma \tag{9}$$

where β and γ are constants. The *detail* and the *visual worth*, as calculated here, are essentially linear functions of size.

Mapping Edges

Straight line edges of the normal view get mapped to straight line edges in the fisheye view automatically when vertices at their end points get mapped. The edges with intermediate bend points can be mapped by mapping each bend point separately. Figure 4 demonstrates the effect of straightforward cartesian transformations on a symmetric graph.

Unfortunately, this straightforward approach does not preserve parallelism between edges. This problem can be circumvented by mapping a very large number of intermediate points on each straight line segment individually. However, mapping a very large a number of points may not be computationally feasible for real time response.

The mapping, however, has the property that all the vertical and horizontal lines remain vertical and horizontal after the transformation. Because of this property, our transformation is ideally suited for graphs with edges consisting of mostly horizontal and vertical line segments, for example VLSI circuits.

DISTORTION

Early users of our prototype system commented that transformations seemed somewhat unnatural, especially when applied to familiar objects, such as maps. Our framework allows us to address this complaint by using domain-specific transformations.

Consider for instance, the non-fisheye view of a map of the United States shown in Figure 6 and a corresponding fisheye view in Figure 7. A more natural fisheye view of such a map might be to distort the map onto a hemisphere. To do so, we developed a transformation based on the polar coordinate system with the origin at the focus (see Figure 5). In this transformation, a point with normal coordinates (r_{norm}, θ) is mapped to the fisheye coordinates (r_{feye}, θ) where

$$r_{feye} = r_{max} \frac{(d+1)\frac{r_{norm}}{r_{max}}}{d\frac{r_{norm}}{r_{max}} + 1} \tag{10}$$

Here, r_{max} is the maximum possible value of r in the same direction as θ. Note that θ remains unchanged by this mapping. Figure 8 shows a resulting fisheye view of the map of the United States. This can be contrasted to Figure 7 which shows a fisheye view of the same outline using the cartesian transformation of Equation 5.

Another factor contributing to the perceived unnaturalness of the fisheye view is that the shapes of vertices remain undistorted and edges remain straight lines (ignoring bend points). We could remedy this by mapping many points on the outline of the vertex, and mapping a large number of intermediate points for the edges, thus allowing the vertices and edges to become curved. However, in our prototype browser, we chose not to do so, in order to achieve real time performance.

THE PROTOTYPE SYSTEM

Our system displays a fisheye view of a user-specified graph, and updates the display in real time as the user moves the focus by dragging with the mouse. Sliders allow the user to control of the value of the distortion factor d in Equation 5, the coefficient c and the exponent e in Equation 7, the vertex scaling factor s also in Equation 7, and a cutoff point at which vertices and their incident edges should no

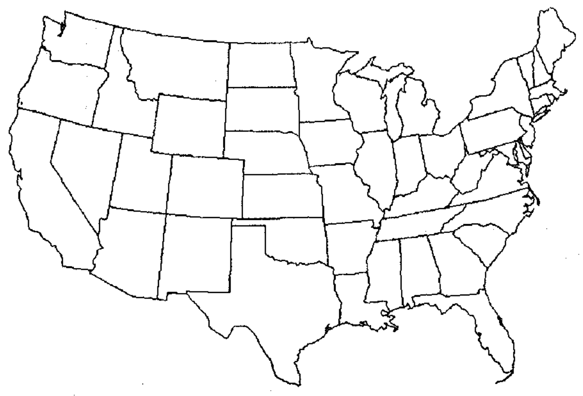


Figure 6: Outline of the United States

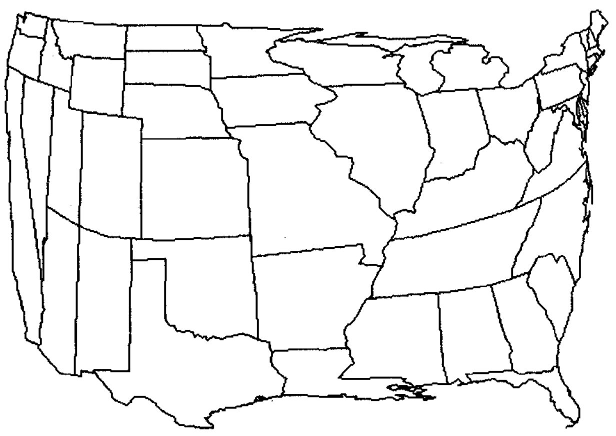


Figure 7: A cartesian transformation of Figure 6. The focus is at the point where Missouri, Kentucky, and Tennessee meet.

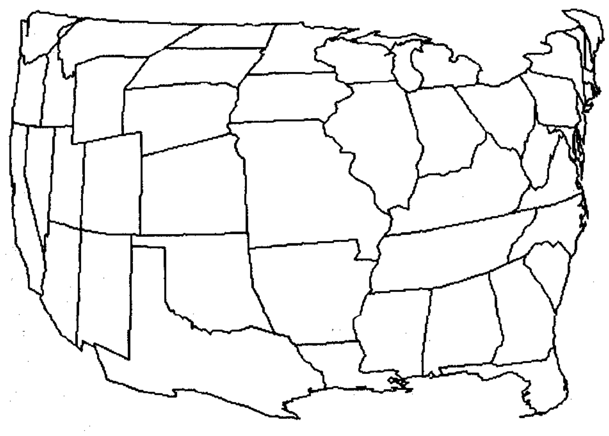


Figure 8: A polar transformation of Figure 6. The focus is at the point where Missouri, Kentucky, and Tennessee meet.

longer be displayed. The coefficient c and the exponent e in Equation 7 control the effect of the API of the vertices on the non-geometric part of the transformation, while d affects the geometric part of the transformation. The combined effect of these parameters on the graph in Figure 1 are illustrated in Figures 2, 12, and 13.

All figures in this paper are screen dumps from our system. To save space, we've cropped the window dressing and user controls from many of the images.

Implementation

The prototype is implemented in an event-driven style. Each time the user moves the mouse while the button is held down, the function $GetFocus$ returns the position of the mouse:

```
loop
  $f$ := GetFocus()
  if $f \neq f_{old}$ then
    foreach $v \in V$
      eval $P_{feye}(v, f), S_{feye}(v, f), DTL_{feye}(v, f)$
    end
    foreach $e \in E$
      if not straightLine($e$) then
        foreach $bp \in bendPoints(e)$
          mapPoint($bp, f$)
        end
      end
    end
    foreach $v \in V$
      eval $VW(v, f)$
    end
    foreach $e \in E$
      if VW($e.v_1$) $\geq$ VWcutoff and
         VW($e.v_2$) $\geq$ VWcutoff then
           repaint edge between $v_1$ and $v_2$
      end
    end
    sort $V$ in order of $VW$
    foreach $v \in V$ in nondecreasing order of $VW$
      if $VW(v) \geq VWcutoff$ then
        repaint vertex $v$
      end
    end
    $f_{old}$ := $f$
  end
```

The system normally ensures that the location of the focus is the same in both normal and fisheye coordinates. However, when the cursor is within the boundary of a vertex, the vertex becomes the focus vertex and the view is not updated until the cursor exits the vertex. Since the size of a vertex in focus is usually large, exiting the focus vertex causes a relatively large shrinkage in the size of the focus vertex and also a relatively large variation in the fisheye view. In particular, since the entry and exit events happen at two different distances from the center of the focus vertex (because at exit-time the size of the vertex is larger than its size at entry-time), without careful coding an exit event causes the most recent focus vertex to shift away by a large distance from the cursor in a jerky motion. One approach to solving this problem is to force the cursor to be positioned just outside the boundary of the most recent focus vertex on each exit event.

Sorting the vertices in order of their visual worth produces a very useful order. First, if the position of two vertices are in conflict, their VW can be used to resolve the conflict in favor of displaying the vertex with higher VW. Second, the order can be used to maintain the real time response of the system, as we shall discuss below.

Response Time

Our prototype system is able to maintain real time response on a DECstation 5000 for graphs of up to about 100 vertices and about 100 horizontal or vertical edges. Computing fisheye views takes an insignificant amount of time compared to the time required for painting. Real time response cannot be maintained for graphs with significantly larger number of vertices and edges. Performance also suffers when the the percentage of edges that are neither horizontal nor vertical is increased.

An alternative "inner loop" is to display "approximate" fisheye views by painting only a fixed number of vertices and edges, irrespective of the size of the graph. Each time there is a new focus, quickly compute the new fisheye view for all vertices, but repaint only those nodes and edges which will give the best approximation to the perfect fisheye view. Nodes with highest change in their VW and nodes with highest current VW are good candidates. One can take a suitable mix of these two types of nodes, as well as all the associated edges. Each update operation will then involve erasing and painting a fixed number of nodes and edges.

System Notes

The prototype is implemented using Modula-3 and Trestle, a portable X-toolkit [8]. This project was the first Trestle application to be written,[1] beyond the handful of small examples in the distribution package. A number of features that we needed for real time animation (e.g., fast double buffering), and aesthetic drawings (e.g., curves and lines of arbitrary thickness) were not functional when the initial prototype was developed during the summer of 1991. We are currently upgrading to the latest release of Trestle.

GENERALIZED FISHEYE VIEWS AND RELATED WORK

Our work follows from the *generalized* fisheye views by Furnas [4, 5]. Furnas gave many compelling arguments describing the advantages of fisheye views, and performed a number of experiments to validate his claims. The essence of Furnas's formalism is the "degree of interest" function for an "object" relative to the "focal point" in some "structure". Our notion of "visual worth" (see Equation 4) is nearly identical to Furnas's degree of interest. The difference is that we

[1] A Modula-2 version of Trestle that doesn't use the X-toolkit has been operational for a number of years at DEC SRC.

have (thus far) described distance as the Euclidean distance separating two vertices in a graph, whereas Furnas defined the distance function as an arbitrary function between two objects in a structure. Our system supports generalized fisheye views by recoding the distance function used explicitly in Equation 4 and implicitly by Equations 1–3.

For instance, consider the graph in Figure 9 (all edges point downwards). The graphical fisheye view of the graph is shown in Figure 10. The API of each vertex is related to its display level (e.g., the root has the highest API of 8, node 33 has an API of 4, and node 86 has an API of 2). The distance between vertices is their Euclidean distance. A vertex is displayed only if its visual worth is above some threshold, and its position, size, and level detail are computed using Equations 1, 2, and 3, respectively. A "generalized" fisheye view of that same graph, with the same focus, is shown in Figure 11. Here, the API is as before, but the distance function not geometrical; it is the length of the shortest path between a vertex and the vertex defining the focus, as proposed by Furnas [5]. Notice that in the generalized fisheye view, each node is either displayed or omitted; there is no explicit way to vary size and and level of detail.

Furnas raised the question of multiple foci [5], but left it unanswered. Our framework can be extended to multiple foci. A simple approach is to divide the screen-space among all the foci (using some criteria), and then apply the transformation independently on each portion of the screen.

Furnas cites a delightful 1973 doctoral thesis by William Farrand [3] as one of the earliest uses of fisheye views of information on a computer screen. The thesis suggests transformations similar to our cartesian and polar transformations, but provides few details.

Last year at CHI '91, Card, Mackinlay, and Robertson presented two views of structured information that have fisheye properties. The *perspective wall* [7] maps a wide 2-dimensional layout into a 3-dimensional visualization. The center panel shows detail, while the two side panels, receding in the distance, show the context. The *cone tree* [9] displays a tree with each node the apex of a cone, and the children of the node positioned around the rim of the cone. The fact that the tree is beautifully rendered in 3D, including shadows and transparency, provides the basic fisheye property of showing local information in detail (because when it is close to the viewer it is large), while also showing the entire context. It would be interesting to experimentally compare cone trees and generalized graphical fisheye views as techniques for visualizing hierarchical information.

It may be fruitful to combine fisheye views with other techniques for viewing extremely large data. For example, related nodes can be combined to form cluster nodes, and the member nodes of a cluster node can be thought of as the detail of the cluster node [2]. The amount of detail to be shown can then be computed using the framework we have presented in this paper. In situations where the information associated with the nodes is very large, one can use fisheye views as navigation tool while the actual information in nodes can be displayed in separate windows.

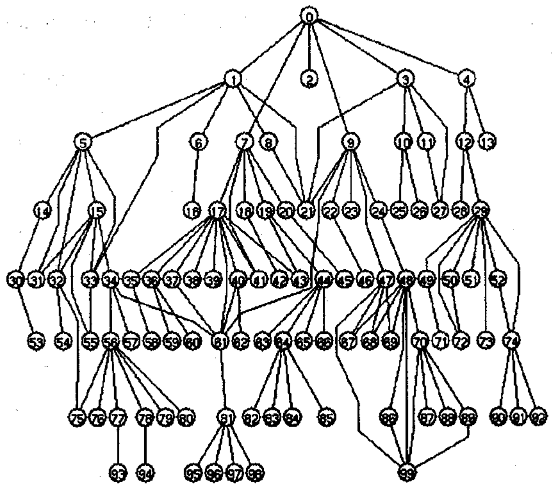


Figure 9: A graph with 100 vertices and 124 edges.

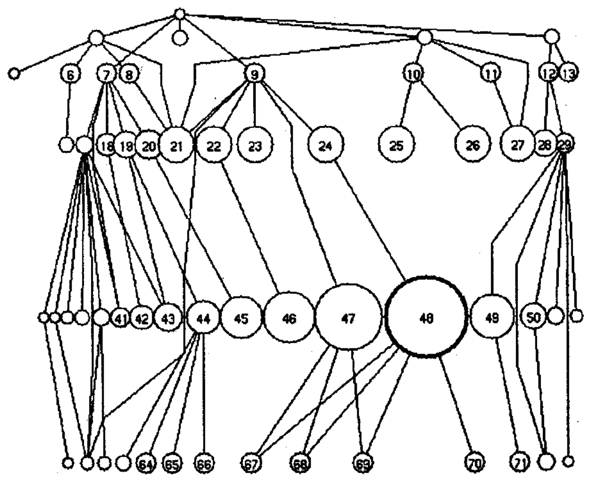


Figure 10: A graphical fisheye view of 9. The focus is on the vertex labeled 48

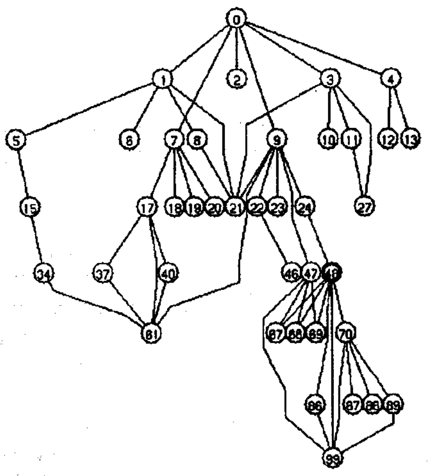


Figure 11: A generalized (non-graphical) fisheye view of 9. The focus is on the vertex labeled 48.

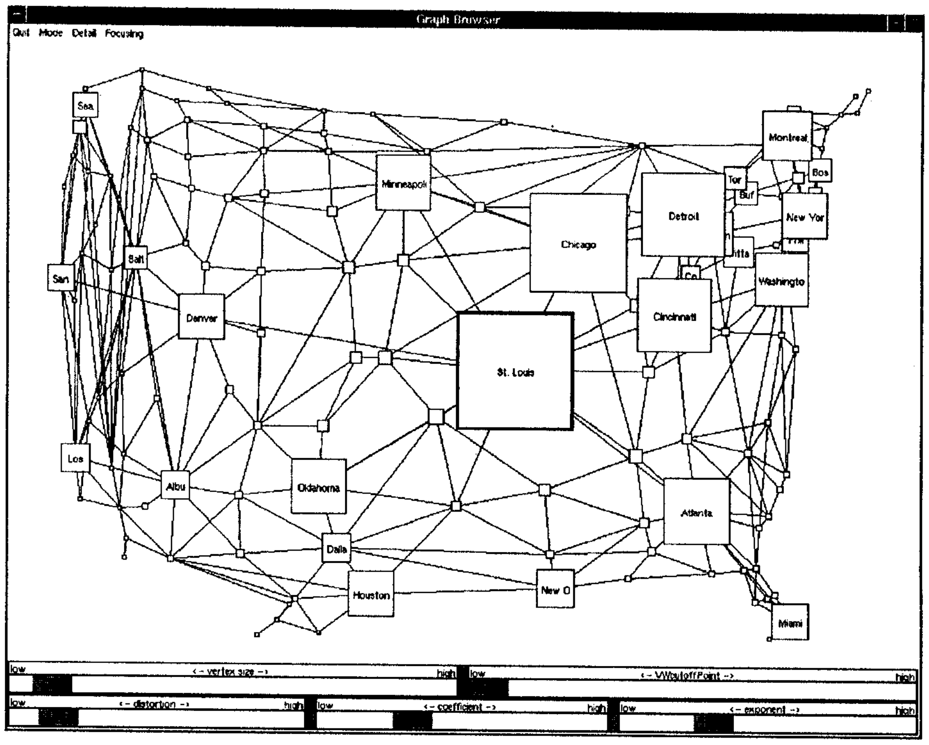


Figure 12: A fisheye view of the graph in Figure 1, $d = 2.38, c = 1.0, e = 1.14, VWcutoff = 0$

SUMMARY

The fisheye view is a promising technique for viewing and browsing structures. Our major contribution is to introduce layout considerations into the fisheye formalism. This includes the position of items, as well as the size and level of detail displayed, as a function of an object's distance from the focus and the object's preassigned importance in the global structure. A second contribution is the notion of a normal coordinate system, thereby allowing layout to be viewed as distortions of some normal structure. As we pointed out, our contributions apply to generalized fisheye views of arbitrary structures (by changing the interpretation of "distance"), in addition to graphs.

It is important to realize that we do not claim that a fisheye view is *the* correct way to display and explore a graph. Rather, it is one of the many ways that are possible. Discovering and quantifying the strengths and weaknesses of fisheye view are challenges for the future.

ACKNOWLEDGMENTS

Jorge Stolfi helped with various ideas concerning geometric transformations. Steve Glassman, Bill Kalsow, Mark Manasse, Eric Muller, and Greg Nelson extricated us from numerous Modula-3 and Trestle entanglements. George Furnas helped to improve the clarity of this presentation considerably.

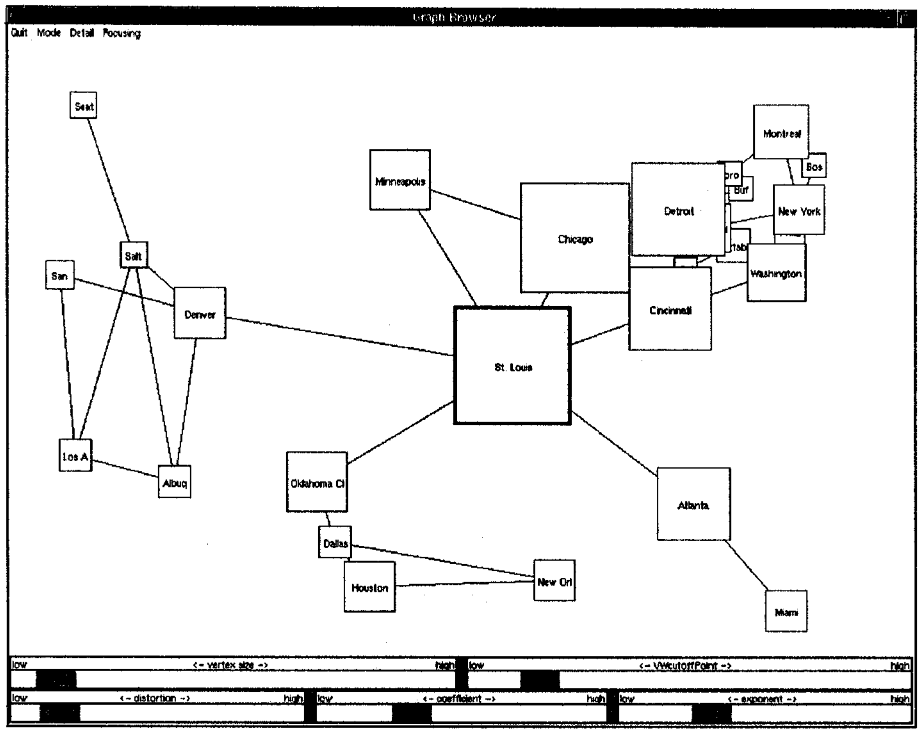


Figure 13: A fisheye view of the graph in Figure 1, $d = 2.38, c = 1.0, e = 1.14, VWcutoff = 0.27$

REFERENCES

[1] Peter Eades and Roberto Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical Report CS-89-90, Department of Computer Science, Brown University, Providence, RI, 1989.

[2] Kim M. Fairchild, Steven E. Poltrok, and George W. Furnas. SemNet: Three-dimensional graphic representations of large knowledge bases. In *Cognitive Science and Its Applications for Human Computer Interaction*, pages 201–233, 1988.

[3] William Augustus Farrand. Information display in interactive design. Ph.D. Thesis, Department of Engineering, UCLA, Los Angeles, CA, 1973.

[4] George W. Furnas. The fisheye view: A new look at structured files. Technical Memorandum 82–11221–22, Bell Laboratories, 1982.

[5] George W. Furnas. Generalized fisheye views. In *Proc. ACM SIGCHI '86 Conf. on Human Factors in Computing Systems*, pages 16–23, 1986.

[6] Tyson R. Henry and Scott E. Hudson. Interactive graph layout. In *Proc. ACM SIGGRAPH, SIGCHI Symposium on User Interface Software and Technology*, pages 55–65, 1991.

[7] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: Detail and context smoothly integrated. In *Proc. ACM SIGCHI '91 Conf. on Human Factors in Computing Systems*, pages 173–179, April 1991.

[8] Greg Nelson, Editor. *Systems Programming with Modula-3*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[9] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone Trees: Animated 3D visualizations of hierarchical information. In *Proc. ACM SIGCHI '91 Conf. on Human Factors in Computing Systems*, pages 189–194, April 1991.